

Proposing a New Foundation of Attack Trees in Monoidal Categories

Harley Eades III

Computer and Information Sciences, Augusta University, Augusta, GA,
heades@augusta.edu

Abstract. TODO

1 Introduction

What do propositional logic, multisets, directed acyclic graphs, source sink graphs (or parallel-series pomsets), Petri nets, and Markov processes all have in common? They are all mathematical models of attack trees – see the references in [9] – but even more than that, they can all be modeled in some form of a symmetric monoidal category¹ [16,2,5,6] – for the definition of a symmetric monoidal category see Appendix A. Taking things a little bit further, monoidal categories have a tight correspondence with linear logic through the beautiful Curry-Howard-Lambek correspondence [1]. This correspondence states that objects of a monoidal category correspond to the formulas of linear logic and the morphisms correspond to proofs of valid sequents of the logic. I propose that attack trees – in many different flavors – be modeled as objects in monoidal categories, and hence, as formulas of linear logic.

The Curry-Howard-Lambek correspondence is a three way relationship:

Categories	\iff	Logic	\iff	Functional Programming
Objects	\iff	Formulas	\iff	Types
Morphisms	\iff	Proofs	\iff	Programs

By modeling attack trees in monoidal categories we obtain a sound mathematical model, a logic for reasoning about attack trees, and the means of constructing a functional programming language for defining attack trees (as types), and constructing semantically valid transformations (as programs) of attack trees. Keep in mind that as stated “semantically valid transformations as programs” is very broad, because this may turn out to be something other than morphisms between attack trees, e.g. it may turn out to be bi-simulation between processes instead.

Linear logic was first proposed by Girard [7] and was quickly realized to be a theory of resources. In linear logic, every hypothesis must be used exactly once. Thus, formulas like $A \otimes A$ and A are not logically equivalent – here \otimes is linear

¹ We provide a proof that the category of source sink graphs is monoidal in Appendix B.

conjunction. This resource perspective of linear logic has been very fruitful in computer science. It has lead to linear logic as being a logical foundation of concurrency where formulas may be considered as processes. This perspective fits modeling attack trees perfectly, because they essentially correspond to concurrent processes. Connecting attack trees to processes has been done before where they have been modeled by event-based models of concurrency like Petri nets and partially-ordered multisets (pomsets) [9,10]. In fact, pomsets is a model in which events (the resources) can be executed exactly once, and thus, has a relationship with linear logic [14]. However, connecting linear logic as a theory of attack trees is novel.

Girard’s genius behind linear logic was that he isolated the structural rules – weakening and contraction – by treating them as an effect and putting them inside a comonad called the of-course exponential denoted $!A$. In fact, $!A \otimes !A$ is logically equivalent to $!A$, and thus, by staying in the comonad we become propositional. This implies that a modal of attack trees in linear logic also provides a model of attack trees in propositional logic, and a combination of the two. It is possible to have the best of both worlds.

In this short paper I introduce a newly funded research project² investigating founding attack trees in monoidal categories, and through the Curry-Howard-Lambek correspondence deriving a new domain-specific functional programming language called Lina for Linear Threat Analysis. I begin by defining an extension of the attack trees given in [9] in Section 2. Then I introduce a new semantics of attack trees in dialectica spaces, which is also a model of full intuitionistic linear logic, in Section 3. The final section, Section 4, discusses Lina and some of the current problems the project seeks to answer.

2 Attack Trees

In this paper I consider an extension of attack trees with sequential composition which are due to Jhavar et al. [9], but one of our ultimate goals is to extend attack trees with even more operators driven by are choice of semantics. The syntax for attack trees is defined in the following definition.

Definition 1. *The following defines the syntax of **Attack Trees** given a set of base attacks $b \in B$:*

$$t ::= b \mid t_1 + t_2 \mid t_1 \sqcup t_2 \mid t_1; t_2 \mid t_1 \otimes t_2 \mid \odot t$$

We denote parallel composition by $t_1 + t_2$, choice between attacks by $t_1 \sqcup t_2$, sequential composition of attacks by $t_1; t_2$, and two new operators called orthocurrence, denoted $t_1 \otimes t_2$, and copy, denoted $\odot t$.

The following rules define the attack tree reduction relation:

² This material is based upon work supported by the National Science Foundation CRII CISE Research Initiation grant, “CRII:SHF: A New Foundation for Attack Trees Based on Monoidal Categories“, under Grant No. 1565557.

$$\begin{array}{c}
\frac{}{(t_1 \text{ op } t_2) \text{ op } t_3 \rightsquigarrow t_1 \text{ op } (t_2 \text{ op } t_3)} \text{ ASSOC} \qquad \frac{}{t_1 \text{ ops } t_2 \rightsquigarrow t_2 \text{ ops } t_1} \text{ SYM} \\
\frac{}{t \sqcup t \rightsquigarrow t} \text{ CHOICE} \qquad \frac{}{\textcircled{C}t \otimes \textcircled{C}t \rightsquigarrow \textcircled{C}t} \text{ COPY} \\
\frac{}{(t_1 \sqcup t_2) + t \rightsquigarrow (t_1 + t) \sqcup (t_2 + t)} \text{ DIST}_1 \qquad \frac{}{(t_1 \sqcup t_2); t \rightsquigarrow (t_1; t) \sqcup (t_2; t)} \text{ DIST}_2
\end{array}$$

where $\text{op} \in \{+, \otimes, ;, \sqcup\}$ and $\text{ops} \in \{+, \otimes, \sqcup\}$. The previous rules can be applied on any well-formed subattack tree, and can be straightforwardly extended into an equivalence relation.

The syntax given in the previous definition differs from the syntax used by Jhawar et al. [9]. First, I use infix binary operations, while they use prefix n -ary operations. However, it does not sacrifice any expressivity, because each operation is associative, and parallel composition, choice, and orthocurrence are symmetric. Thus, we can embed Jhawar et al.'s definition of attack trees into the ones defined here.

The second major difference is that I denote parallel composition with an operator that implies we can think of it as a disjunction, but Jhawar et al. and others in the literature seem to use an operator that implies that we can think of it as a conjunction. The semantics, however, tells us that it is really a disjunction. The parallel operation defined on source sink graphs defined by Jhawar et al. [9] can be proven to be a coproduct – see Appendix B – and coproducts categorically model disjunctions. Furthermore, parallel composition is modeled by multiset union in the multiset semantics, but we can model this as a coproduct. Lastly, the semantics I give in the next section models parallel composition as a coproduct. Thus, I claim that an operator that reflects this is for the better.

The third difference is that I denote the choice between executing attack t_1 or attack t_2 , but not both, by $t_1 \sqcup t_2$ instead of using a symbol that implies that it is a disjunction. This fits very nicely with the semantics of Jhawar et al., where they collect the attacks that can be executed into a set. The semantics I given in the next section models choice directly.

The forth, and final, difference is that we extend the syntax with two new operators called orthocurrence and copy. The attack $t_1 \otimes t_2$ states that t_1 interacts with the attack t_2 in the sense that processes interact. Modeling interacting attacks allows for the more refined modeling of security critical systems, for example, it can be used to bring social engineering into the analysis where someone communicates malicious information or commands to a unsuspecting party. As a second example, one could take over a workstation on a network and funnel malicious traffic through it onto the internal network. Orthocurrence stems from process algebra, and for more examples, and a brief history of orthocurrence see [13]. I conjecture that it should be possible to extend Jhawar et al.'s model of attack trees to include orthocurrence due to the relationship their model has with pomsets which is where orthocurrence actually originated.

The attack $\textcircled{C}t$ indicates that attack t can be copied and contracted. For example, $\textcircled{C}t \otimes \textcircled{C}t$ is equivalent to $\textcircled{C}t$. Thus, the attack trees given here can

treat attack trees as processes/resources that cannot be freely copied and deleted, but also as propositions that can be, but even further, these two perspectives can be mixed. Semantically, $\odot t$ is equivalent to the of-course exponential from linear logic mentioned in the introduction.

The reduction rules are a slightly extended version of equivalences given in Jhaware et al. [9] – Theorem 1. The main difference is the COPY rule which allows copies made by the copy operator to be contracted.

3 Semantics of Attack Trees in Dialectica Spaces

I now introduce a new semantics of attack trees that connects their study with two new perspectives that could highly impact future research in attack trees: intuitionistic linear logic and process calculi. Note that every construction and proof given in this section with the exception of sequential composition has been formalized in the proof assistant Agda³. The semantics is based on the notion of a dialectica space:

Definition 2. A *dialectica space* is a triple (A, Q, δ) where A and Q are sets and $\delta : A \times Q \rightarrow 2$ is a relation.

Dialectica spaces can be seen as the intuitionistic cousin [4] of Chu spaces [12]. The latter have been used extensively to study process algebra and as a model of classical linear logic, while dialectica spaces and their morphisms form a model of full intuitionistic linear logic [3]. I will use the intuitions often used when explaining Chu spaces to explain dialectica spaces, but it should be known that these intuitions are due to Pratt and Gupta [8]. However, verifying that the Chu space construction of choice and sequential composition works in dialectica spaces is novel and so is the application of this semantics to attack trees.

Intuitively, we can think of a dialectica space, (A, Q, δ) , as a process where A is the set of actions the process will execute, Q is the set of states the process can enter, and for $a \in A$ and $q \in Q$, $\delta(a, q)$ indicates whether action a can be executed in state q .

Dialectica spaces form the objects of a category called $\text{Dial}_2(\text{Sets})$ which forms one of the earliest models of linear logic – originally due to de Paiva [3] – and is the first model of intuitionistic linear logic that contains every linear operator. The definition $\text{Dial}_2(\text{Sets})$ requires the notion of a morphism between dialectica spaces, but due to space we do not define it here; see the formal development for the definition.

The interpretation of attack trees into dialectica spaces, each definition is equivalent to the construction on Chu spaces [8], requires the construction of each operation on dialectica spaces:

³ The complete formalization can be found at <https://github.com/heades/dialectica-spaces> which is part of a general library for working with dialectica spaces in Agda developed with Valeria de Paiva.

Parallel Composition. Suppose $\mathcal{A} = (A, Q, \alpha)$ and $\mathcal{B} = (B, R, \beta)$ are two dialectica spaces. Then we can construct the (coproduct) dialectica space $\mathcal{A} + \mathcal{B} = (A + B, Q \times R, \alpha + \beta)$ where $A + B$ is the disjoint union of A and B , and $\alpha + \beta : (A + B) \times (Q \times R) \rightarrow 2$ is defined by $(\alpha + \beta)(i, x) = \alpha(i, x)$ if $i \in A$, but $(\alpha + \beta)(i, x) = \beta(i, x)$ if $i \in B$. Thus, from a process perspective we can see that $\mathcal{A} + \mathcal{B}$ executes either an action of \mathcal{A} or an action of \mathcal{B} , but also potentially both.

Choice. Suppose $\mathcal{A} = (A, Q, \alpha)$ and $\mathcal{B} = (B, R, \beta)$ are two dialectica spaces. Then we can construct the dialectica space $\mathcal{A} \sqcup \mathcal{B} = (A + B, Q + R, \alpha \sqcup \beta)$ where $\alpha \sqcup \beta : (A + B) \times (Q + R) \rightarrow 2$ is defined by $(\alpha \sqcup \beta)(i, j) = \alpha(i, j)$ if $i \in A$ and $j \in Q$, $(\alpha \sqcup \beta)(i, j) = \beta(i, j)$ if $i \in B$ and $j \in R$, otherwise $(\alpha \sqcup \beta)(i, j) = 0$. Thus, from a process perspective we can see that $\mathcal{A} \sqcup \mathcal{B}$ executes either an action of \mathcal{A} or an action of \mathcal{B} , but not both. Since the actions and states of $\mathcal{A} \sqcup \mathcal{B}$ are disjoint unions it is pretty easy to show that choice is symmetric and associative, but it is not a coproduct, because it is not possible to define the corresponding injections.

Sequential Composition. Modeling sequential conjunction requires we extend the notion of dialectic space with the ability to determine which states of the space are initial and which are final, but this is straightforward. Suppose $\mathcal{A} = (A, Q, \alpha)$ and $\mathcal{B} = (B, R, \beta)$ are two dialectica spaces. Then we can construct the dialectica space $\mathcal{A}; \mathcal{B} = (A + B, Z, \alpha; \beta)$ where $Z = \{(q_1, q_2) \in Q \times R \mid q_1 \text{ is final in } \mathcal{A} \text{ or } q_2 \text{ is initial in } \mathcal{B}\}$, and $\alpha; \beta : (A + B) \times Z \rightarrow 2$ is defined by $(\alpha; \beta)(i, (q, r)) = \alpha(i, q)$ if $i \in A$, $(\alpha; \beta)(i, (q, r)) = \beta(i, r)$ if $i \in B$. Thus, from a process perspective we can see that $\mathcal{A}; \mathcal{B}$ will first execute the actions of \mathcal{A} and then once in a final state it will begin executing actions of \mathcal{B} . It can be shown that sequential composition is a non-symmetric associative operation.

Orthocurrence. Suppose $\mathcal{A} = (A, Q, \alpha)$ and $\mathcal{B} = (B, R, \beta)$ are two dialectica spaces. Then we can construct the dialectica space $\mathcal{A} \otimes \mathcal{B} = (A \times B, (B \rightarrow Q) \times (A \rightarrow R), \alpha \otimes \beta)$ where $B \rightarrow Q$ and $A \rightarrow R$ denote function spaces, and $\alpha \otimes \beta : (A \times B) \times ((B \rightarrow Q) \times (A \rightarrow R)) \rightarrow 2$ is defined by $(\alpha \otimes \beta)((a, b), (f, g)) = \alpha(a, f(b)) \wedge \beta(b, g(a))$. From a process perspective the actions of $\mathcal{A} \otimes \mathcal{B}$ are actions from \mathcal{A} and actions of \mathcal{B} , but the states are pairs of maps $f : B \rightarrow Q$ and $g : A \rightarrow R$ from actions to states. This is the point of interaction between the processes. It is possible to show that orthocurrence is a symmetric monoidal bi-functor which is associative, and has an identity, but for the details see the formal development.

Copying. Suppose $\mathcal{A} = (A, Q, \alpha)$ is a dialectica space. Then $\odot \mathcal{A} = (A, A \rightarrow Q^*, \alpha^*)$ where Q^* denotes the free monoid with carrier Q . Copying defines a functor such that if we have a morphism $f : \mathcal{B} \rightarrow \mathcal{C}$, then we obtain a morphism $\odot f : \odot \mathcal{B} \rightarrow \odot \mathcal{C}$. Furthermore, $\odot : \text{Dial}_2(\text{Sets}) \rightarrow \text{Dial}_2(\text{Sets})$ defines a comonad on the category of dialectica spaces, and thus, we have dialectica morphisms $\varepsilon : \odot \mathcal{A} \rightarrow \mathcal{A}$ and $\delta : \odot \mathcal{A} \rightarrow \odot \odot \mathcal{A}$ satisfying the usual diagrams. Therefore, we have an isomorphism $(\odot \mathcal{A} \otimes \odot \mathcal{A}) \cong \odot \mathcal{A}$. This implies that under \odot we escape to propositional logic.

At this point it is straightforward to define an interpretation $\llbracket t \rrbracket$ of attack trees into $\text{Dial}_2(\text{Sets})$. Soundness with respect to this model would correspond to if $t_1 \rightsquigarrow t_2$, then $\llbracket t_1 \rrbracket \cong \llbracket t_2 \rrbracket$ where the latter takes place in $\text{Dial}_2(\text{Sets})$ for some suitable equivalence \cong between objects. Naturally, one might choose isomorphism, but this does not hold, because the reduction rules COPY, DIST₁, and DIST₂ do not hold up to isomorphism of objects, but we conjecture if we take \cong to be bi-simulation as defined for Chu spaces – see p. 63 of Gupta [8] – then we obtain the proper equivalences.

This semantics can be seen as a generalization of some existing models. Multisets, pomsets, and Petri nets can all be modeled by dialectica spaces [2,8]. However, there is a direct connection between dialectica spaces and linear logic which leads to a logical theory of attack trees.

4 Lina: A Domain Specific PL for Threat Analysis

The second major part of this project is the development of a statically-typed polymorphic domain-specific linear functional programming language for specifying and reasoning about attack trees called Lina for Linear Threat Analysis. Lina will consist of a core language and a surface language. The core language will include a decidable type checker using term annotations on types; note that type checking becomes undecidable in the presence of parametric polymorphism [17]. Programming with annotations can be very cumbersome, and so the surface language will use local type inference [11] to alleviate some of the burden from annotations. However, the surface language will provide further conveniences in the form of automation, to be used with labeled attack trees, and graphical representations of attack trees based on the various graphical languages used in category theory [15]. Thus, the security specialist will not be required to program directly in Lina, but instead will only be required to use a graphical interface to construct attack trees and prove properties about them.

There are two main questions that still must be answered in order for all this to work. First, linear logic generally does not include operators like choice and sequential composition, and hence, in order to build Lina we must figure out how to add these operators to linear logic. It should be enough to add a new non-commutative monoidal operator for sequential conjecture to linear logic along with a new symmetric monoidal operator for choice. It seems the former has been done in classical linear logic [14], but it remains to be seen if it can be done in intuitionistic linear logic.

Second, programs are seen as proofs/dialectica space morphisms, but we know that the morphisms of $\text{Dial}_2(\text{Sets})$ do not capture all of the structure of the attack trees, and hence, we must move to bi-simulation of dialectica spaces to obtain a sound model of attack trees. So one important question we must answer is do all of isomorphisms between dialectica spaces in $\text{Dial}_2(\text{Sets})$ lift to equivalences using bi-simulation? If yes, then we may consider programs as bi-simulations, and hence, obtain the proper structure to soundly model attack trees.

5 Conclusion

The project described here is to first develop the semantics of attack trees (Section 2) in dialectica spaces (Section 3), a model of full intuitionistic linear logic, and then exploiting – and extending – the Curry-Howard-Lambek correspondence to develop a new functional programming language called Lina (Section 4) to be used to develop a new tool to conduct threat analysis using attack trees.

This tool will include the ability to design and formally reason about attack trees using interactive theorem proving.

References

1. Michael Barr. *-autonomous categories and linear logic. *Mathematical Structures in Computer Science*, 1:159–178, 7 1991.
2. Carolyn Brown, Doug Gurr, and Valeria Paiva. A linear specification language for petri nets. *DAIMI Report Series*, 20(363), 1991.
3. Valeria de Paiva. Dialectica categories. In J. Gray and A. Scedrov, editors, *Categories in Computer Science and Logic*, volume 92, pages 47–62. American Mathematical Society, 1989.
4. Valeria de Paiva. Dialectica and chu constructions: Cousins? *Theory and Applications of Categories*, 17(7):127–152, 2006.
5. Marcelo Fiore and Marco Devesas Campos. *Computation, Logic, Games, and Quantum Foundations. The Many Facets of Samson Abramsky: Essays Dedicated to Samson Abramsky on the Occasion of His 60th Birthday*, chapter The Algebra of Directed Acyclic Graphs, pages 37–51. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
6. Luisa Francesco Albasini, Nicoletta Sabadini, and Robert F. C. Walters. The compositional construction of markov processes. *Applied Categorical Structures*, 19(1):425–437, 2010.
7. Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1 – 101, 1987.
8. Vineet Gupta. *Chu Spaces: a Model of Concurrency*. PhD thesis, Stanford University, 1994.
9. Ravi Jhawar, Barbara Kordy, Sjouke Mauw, Saúl A. Radomirović, and Rolando Trujillo-Rasua. Attack trees with sequential conjunction. In Hannes Federrath and Dieter Gollmann, editors, *ICT Systems Security and Privacy Protection*, volume 455 of *IFIP Advances in Information and Communication Technology*, pages 339–353. Springer International Publishing, 2015.
10. Sjouke Mauw and Martijn Oostdijk. Foundations of attack trees. In DongHo Won and Seungjoo Kim, editors, *Information Security and Cryptology - ICISC 2005*, volume 3935 of *Lecture Notes in Computer Science*, pages 186–198. Springer Berlin Heidelberg, 2006.
11. Benjamin C. Pierce and David N. Turner. Local type inference. *ACM Trans. Program. Lang. Syst.*, 22(1):1–44, January 2000.
12. Vaughan Pratt. Chu spaces. Notes for the School on Category Theory and Applications University of Coimbra, July 1999.
13. Vaughan R. Pratt. Orthocurrence as both interaction and observation. In *In Proc. Workshop on Spatial and Temporal Reasoning*, 2001.
14. Christian Retoré. *Typed Lambda Calculi and Applications: Third International Conference on Typed Lambda Calculi and Applications TLCA '97 Nancy, France, April 2–4, 1997 Proceedings*, chapter Pomset logic: A non-commutative extension of classical linear logic, pages 300–318. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.
15. Peter Selinger. A survey of graphical languages for monoidal categories. *ArXiv e-prints*, August 2009.
16. A Tzouvaras. The linear logic of multisets. *Logic Journal of IGPL*, 6(6):901–916, 1998.

17. J.B. Wells. Typability and type checking in system f are equivalent and undecidable. *Annals of Pure and Applied Logic*, 98(1&3):111 – 156, 1999.

A Symmetric Monoidal Categories

This appendix provides the definitions of both categories in general, and, in particular, symmetric monoidal closed categories. We begin with the definition of a category:

Definition 3. A *category*, \mathcal{C} , consists of the following data:

- A set of objects \mathcal{C}_0 , each denoted by A, B, C , etc.
- A set of morphisms \mathcal{C}_1 , each denoted by f, g, h , etc.
- Two functions **src**, the source of a morphism, and **tar**, the target of a morphism, from morphisms to objects. If $\text{src}(f) = A$ and $\text{tar}(f) = B$, then we write $f : A \rightarrow B$.
- Given two morphisms $f : A \rightarrow B$ and $g : B \rightarrow C$, then the morphism $f;g : A \rightarrow C$, called the composition of f and g , must exist.
- For every object $A \in \mathcal{C}_0$, there must exist a morphism $\text{id}_A : A \rightarrow A$ called the identity morphism on A .
- The following axioms must hold:
 - (Identities) For any $f : A \rightarrow B$, $f; \text{id}_B = f = \text{id}_A; f$.
 - (Associativity) For any $f : A \rightarrow B$, $g : B \rightarrow C$, and $h : C \rightarrow D$, $(f;g);h = f;(g;h)$.

Categories are by definition very abstract, and it is due to this that makes them so applicable. The usual example of a category is the category whose objects are all sets, and whose morphisms are set-theoretic functions. Clearly, composition and identities exist, and satisfy the axioms of a category. A second example is preordered sets, (A, \leq) , where the objects are elements of A and a morphism $f : a \rightarrow b$ for elements $a, b \in A$ exists iff $a \leq b$. Reflexivity yields identities, and transitivity yields composition.

Symmetric monoidal categories pair categories with a commutative monoid like structure called the tensor product.

Definition 4. A *symmetric monoidal category (SMC)* is a category, \mathcal{M} , with the following data:

- An object I of \mathcal{M} ,
- A bi-functor $\otimes : \mathcal{M} \times \mathcal{M} \rightarrow \mathcal{M}$,
- The following natural isomorphisms:

$$\begin{aligned}\lambda_A &: I \otimes A \rightarrow A \\ \rho_A &: A \otimes I \rightarrow A \\ \alpha_{A,B,C} &: (A \otimes B) \otimes C \rightarrow A \otimes (B \otimes C)\end{aligned}$$

- A symmetry natural transformation:

$$\beta_{A,B} : A \otimes B \rightarrow B \otimes A$$

– Subject to the following coherence diagrams:

$$\begin{array}{ccc}
((A \otimes B) \otimes C) \otimes D & \xrightarrow{\alpha_{A,B,C} \otimes \text{id}_D} & (A \otimes (B \otimes C)) \otimes D \\
\downarrow \alpha_{A \otimes B, C, D} & & \downarrow \alpha_{A, B \otimes C, D} \\
(A \otimes B) \otimes (C \otimes D) & & \\
\downarrow \alpha_{A, B, C \otimes D} & & \\
A \otimes (B \otimes (C \otimes D)) & \xleftarrow{\text{id}_A \otimes \alpha_{B, C, D}} & A \otimes ((B \otimes C) \otimes D)
\end{array}$$

$$\begin{array}{ccccc}
(A \otimes B) \otimes C & \xrightarrow{\alpha_{A,B,C}} & A \otimes (B \otimes C) & \xrightarrow{\beta_{A, B \otimes C}} & (B \otimes C) \otimes A \\
\downarrow \beta_{A,B} \otimes \text{id}_C & & & & \downarrow \alpha_{B,C,A} \\
(B \otimes A) \otimes C & \xrightarrow{\alpha_{B,A,C}} & B \otimes (A \otimes C) & \xrightarrow{\text{id}_B \otimes \beta_{A,C}} & B \otimes (C \otimes A)
\end{array}$$

$$\begin{array}{ccc}
(A \otimes I) \otimes B & \xrightarrow{\alpha_{A,I,B}} & A \otimes (I \otimes B) \\
\searrow \rho_A & & \swarrow \lambda_B \\
& A \otimes B &
\end{array}$$

$$\begin{array}{ccc}
A \otimes B & & \\
\downarrow \beta_{A,B} & \searrow \text{id}_{A \otimes B} & \\
B \otimes A & \xrightarrow{\beta_{B,A}} & A \otimes B
\end{array}$$

$$\begin{array}{ccc}
I \otimes A & \xrightarrow{\beta_{I,A}} & A \otimes I \\
\searrow \lambda_A & & \swarrow \rho_A \\
& A &
\end{array}$$

B Source Sink Graphs are Symmetric Monoidal